Intro to Convolutional Networks

AW

Lecture Overview

1. **Extracting Objects in Images**

2. **Basic Structure and Intuition**

3. **Convolution Operation**

Extracting Features from Pictures

- Goal: to detect an object in an image (e.g. a ball)
  - Whatever method is used to recognize objects should not be concerned with the precise location of the object in the image.
- System should exploit this idea of *spatial invariance* to learn useful representations with fewer parameters!
- To implement the idea that we can start 'moving' parts of the image in early layers. Then the network should
  1. Respond similarly to the same patch, regardless of where it appears in the image (this is *translation invariance* idea).
  2. Focus on local regions, without regard for the contents of the image in distant regions (this is the *locality* principle).
  3. Each early layer should enhance a feature (aka *feature mapping*).
- In later layers local representations learned in early layers should be aggregated to make predictions at the whole image level.

What is Transitional Invariance?

What is it really in a computer?

1. An image is a set of pixels so on input we have a matrix X and object (say a ball) is located between some positions in rows $i_s < i < i_e$, and $j_s < j < j_e$. So when we move the picture, the sub-matrix $[X]_{i_s<i<i_e}^{j_s<j<j_e}$ moves too. So that the ball is now located in $[H]_{t_s<i'<t_e}^{r_s<j'<r_e}$ in the new picture H In fact there are for some numbers (a, b) we have $j_s - b = r_s < j' < r_e = j_e - b$, and $i_s - a = t_s < i' < t_e = i_e - a$
2. We want to make X input of layer of the Convolutional Neural Network (CNN) and H to be output of the next layer Then basically to transfer a ball and separate it we just need umbers $w_{i,j,i',j'} = \begin{cases} 1 \text{ if } i' = i - a \text{ and } j' = j - b \\ 0 \text{ otherwise} \end{cases}$ such that $[\mathrm{H}]_{i',j'} = w_{i,j,i',j'} \cdot [X]_{ij}$.

What is Feature Extraction?

1. If we add together the pixel values in the sub-matrix $[X]_{i_s<i<i_e}^{j_s<j<j_e}$ within a ball the value of the sum will be bigger than if we take any submatrix of the background and do the same. Moreover if we center the submatrix at the ball center (i_0, j_0) and add all values and do the same for a point (i, j) on the periphery of the ball the sum at the center will be bigger than on periphery. If we not only add points but enhance them with weights – the further from center the lower the weights- we are going to separate the ball even more.
2. How can we use this fact? Take am image of the ball (*kernel*) with weights decreasing from the center and start moving it around the picture multiplying the picture with filter pixel-by-pixel and summing up. When we position the filter image at the center of the ball we are going to get highest response!

Transitional Invariance + Feature Extraction

3. Let as combine it with transitional invariance:
   - Let now $w_{i,j,k,l}$ be any positive numbers (e.g. image of the ball on the noise background shifted around: pixel k, l moved to position i, j and multiplied by weight $w_{i,j,k,l}$), so kernel is 4D tensor $\boldsymbol{W}$ with elements $[\boldsymbol{W}]_{ijkl}$
   - Let a new output of a hidden layer be (picture) H be defined as follows: $H_{ij} = \sum_k \sum_l [\boldsymbol{W}]_{ijkl} [X]_{kl}$. We want these to be defined for any values of indexes – positive and negative so let's just assume $[\boldsymbol{W}]_{ijkl} = 0 = [X]_{kl}$ for i, j, k, l such that it is not naturally defined in our pictures.
   - Since only k, l are indexed we can just change indexing setting $i = t - k$ and $j = r - l$ and write instead

$$H_{tr} = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} [\boldsymbol{V}]_{t-k,r-l,k,l} [X]_{k,l}$$

   where $[\boldsymbol{W}]_{i,j,k,l} = [\boldsymbol{V}]_{t-k,r-l,k,l}$.

More on Feature Extraction Treatment

- If weights in tensor $\boldsymbol{V}$ are the same for all positions (k, l), i.e. weights depend only on the output neuron position

  $v(a, b, k, l) = v(a, b)$ for all a, b then instead of

  $$H_{tr} = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} [\boldsymbol{V}]_{t-k,r-l,k,l} [X]_{k,l}$$

  we get

  $$H_{t,r} = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} [V]_{t-k,r-l} [X]_{k,l}$$

  where V is a matrix not 4-d tensor. This expression is discrete convolution denoted by $H = V * X$

- Starting with the $H_{tr} = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} [V]_{t-k,r-l} [X]_{k,l}$ and changing indexes from k, l to $i = t - k$ and $j = r - l$ and using the fact that l changes from $-\infty$ to ∞ we obtain

  $$H_{tr} = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} [V]_{i,j} [X]_{t-i,r-j}$$

What is Locality?

- We believe that we should not have to look very far away from location (i_0, j_0) in order to obtain relevant information to assess what is going on at $[H]_{i_0 j_0}$. This means that outside some range $|i| < \Delta$ or $|r| < \Delta$, we should set $[V]_{a,b} = 0$. Equivalent way to write it is

$$H_{tr} = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} [V]_{ij}[X]_{t-i,r-j}$$
$$= \sum_{i=-\Delta}^{\Delta} \sum_{j=-\Delta}^{\Delta} [V]_{ij}[X]_{t-i,r-j}$$
$$= \sum_{i=-\Delta}^{\Delta} \sum_{j=-\Delta}^{\Delta} [V]_{ij}[X]_{t+i,r+j}$$

r

- Consider again a ball kernel. We apply the same filter regardless of position of the center so its $\boldsymbol{V}$ does not depend on t, r so again we can use a 2D matrix for weights centered around t_0, r_0 that we can write

$$H_{t_0 r_0} = \sum_{i=-\Delta}^{\Delta} \sum_{j=-\Delta}^{\Delta} [V]_{ij}[X]_{t_0+i,r_0+j}$$

Algorithm for Convolution Computation

Let X (image) be p × q matrix, and let V (kernel) be $2m + 1 \times 2n + 1$ matrix indexed from $(-n, -m)$ to (n, m)

For each row, $\boldsymbol{i} = 1$ to $\boldsymbol{p}$

  For each column, $\boldsymbol{j} = 1$ to $\boldsymbol{p}$ # Process entry (i, j)

    $g(i, j) = 0$ # Initialize output

    For each kernel row, $\boldsymbol{k} = -\boldsymbol{m}$ to $\boldsymbol{m}$

      For each kernel column, $\boldsymbol{l} = -\boldsymbol{n}$ to n

        $g(i, j) = g(i, j) + X(i + k, j + l)V(k, l)$

Lecture Overview

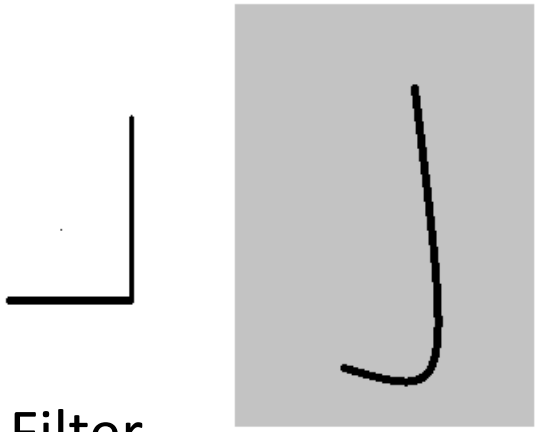
1. Extracting Objects in Images

2. Mock example

3. Basic Structure and Intuition

Picture and Filter

- Suppose we want to find right angle in a picture



Filter

Picture

The filter and digitized picture are:

V

0	**2**	**0**
0	2	0
2	2	0

X

2	2	2	2
2	2	4	2
2	3	4	2
2	3	4	2

Example of Convolution Computed

$$H_{00} = 0+0+0+0+4+0+0+4+0 = 8$$
$$H_{01} = 0+0+0+0+4+0+4+4+0 = 12$$
$$H_{02} = 0+0+0+0+4+0+4+8+0 = 16$$
$$H_{03} = 0+0+0+0+4+0+8+4+0 = 16$$
$$H_{10} = 0+4+0+0+4+0+0+4+0 = 12$$
$$H_{11} = 0+4+0+0+4+0+4+6+0 = 18$$
$$H_{12} = 0+4+0+0+8+0+6+8+0 = 26$$
$$H_{13} = 0+4+0+0+4+0+8+4+0 = 20$$
$$H_{20} = 0+4+0+0+4+0+0+4+0 = 12$$
$$H_{21} = 0+4+0+0+6+0+4+6+0 = 20$$
$$H_{22} = 0+8+0+0+8+0+6+8+0 = 30$$
$$H_{23} = 0+4+0+0+4+0+8+4+0 = 20$$
$$H_{30} = 0+4+0+0+4+0+0+0+0 = 8$$
$$H_{31} = 0+6+0+0+6+0+0+0+0 = 12$$
$$H_{32} = 0+8+0+0+8+0+0+0+0 = 16$$
$$H_{33} = 0+4+0+0+4+0+0+0+0 = 8$$

0	**2**	**0**
0	2	0
2	2	0

2	2	2	2
2	2	4	2
2	3	4	2
2	3	4	2

8	12	16	16
12	18	26	20
12	20	30	20
8	12	16	8

Convolution and Cross-Correlation

What is in fact used is cross-correlation, not convolution. Here is the difference

0	**2**	**0**
0	2	0
2	2	0

Convolution Kernel

To achieve the same result with as with this convolution kernel we need cross-correlation kernel that is flipped relative the center of application moving (i, j) to $(-i, -j)$:

0	**2**	**2**
0	2	0
0	2	0

Respective Cross-Correlation Kernel

This is because on a finite interval the kernel flips (for simplicity I am writing formula in one variable) :

convolution: $x[n] * h[n] = \sum_{k=-\Delta}^{\Delta} h[k]x[n-k]$

cross-correlation: $corr(x[n], h[n]) = \sum_{k=-\Delta}^{\Delta} h[k]x[n+k]$

Lecture Overview

1. Extracting Objects in Images

2. Mock example

3. Basic Structure and Intuition

Birds Eye View

- Convolutional neural networks are *domain-aware* neural networks.
  - The structure of the neural network encodes domain specific information.
    - Specifically designed for images.
    - Images have length, width, and a *depth* corresponding to the number of color channels (typically 3: RGB).
- All layers are spatially structured with length, width, and depth.
- Basic idea: Replace matrix multiplication in neural nets with convolution – will explain just a little bit later
  - Everything else stays the same
  - E.g. maximum likelihood loss
  - Back-propagation, etc.

Basic Architecture

- Most layers have length, width, and depth.
- The depth is 3 for color images, 1 for grayscale, and an arbitrary value for hidden layers.
- Three operations are convolution, max-pooling, and ReLU.
  - The convolution operation is analogous/plays the same role to the matrix multiplication in a conventional network:
    - Take an input, produce an output (hidden layer)
  - “Deconvolution” is analogous to multiplication by transpose of a matrix:
    - Used to back-propagate error from output to input
  - Weight gradient computation
    - Used to backpropagate error from output to weights
    - Accounts for the parameter sharing

Technical Idea of Convolutional Network

- Suppose we want to learn features present in images (noses, eyes, etc.)
- Images are pixel frames, i.e. matrices. Process them locally, i.e. by small parts:
  1. Flatten matrix to a vector. Set window (filter) size and initial point to 0
  2. Until no more entries remain do
     1. Give locality window as input to a single neuron.
     2. Slide with a window of locality along the vector by adding 1 to initial point
  3. Output the results of created neuron into a vector.
- The output vector gives input to a next convolutional layer that is arranged in exactly the same way.
- Neurons that take the vector are first convolutional layer, neurons that take output of the first convolutional layer as input are second convolutional layer, etc.

Illustration of CNN idea

Example:

- Image $I \rightarrow 3 \times 3$ pixel matrix.
- Window is of size 4, i.e. neurons take 4 inputs. Kernel is a 4 vector.
- The flattened vector is $(I_{11} I_{12} \dots I_{31} \dots I_{33})^T$ flattened vector will have an array of values with repetitions
- The first a neuron takes as input $I_{11}, \dots, I_{21}$, the second $I_{12}, \dots, I_{22}$, etc.
- There are total of 6 neurons, so the output will be 6 dimensional vector.

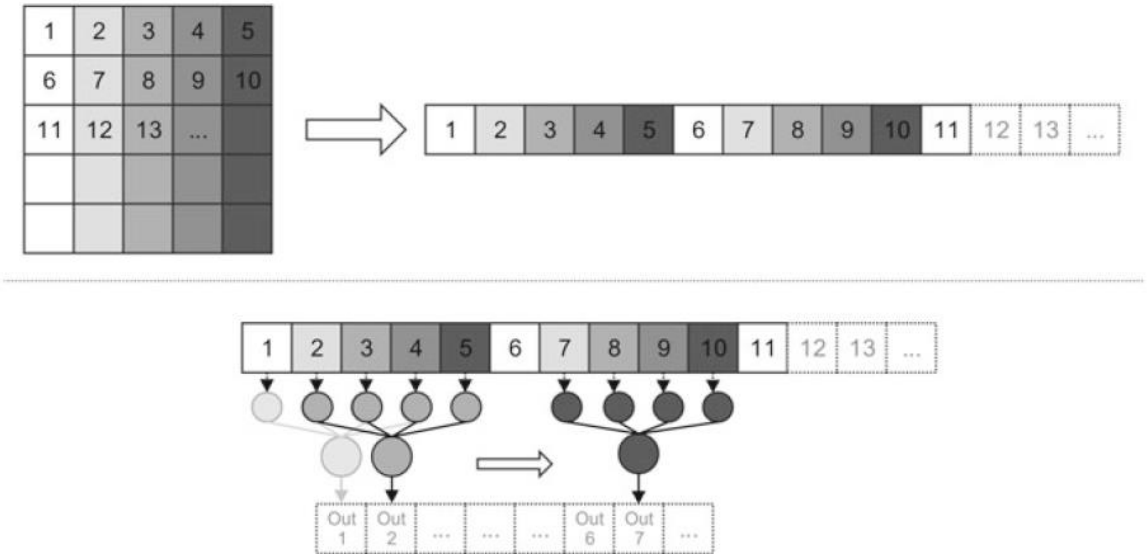
This is 1-D (or temporal) convolutional layer.

Could do same with 2-d or 3-d window

Illustration of CNN idea (continued)

Example:

- We may want to have same dimension for each layer. Then 13 inputs are needed. In this case we *pad* the input vector with 0's on both sides (position numbers of the vector are in the bottom row)
$\begin{pmatrix} 0, & 0, I_{11} \ldots, I_{33}, & 0, & 0, & 0 \\ -1, 0, 1, & \ldots & 9, & 10, & 11, 12 \end{pmatrix}$.



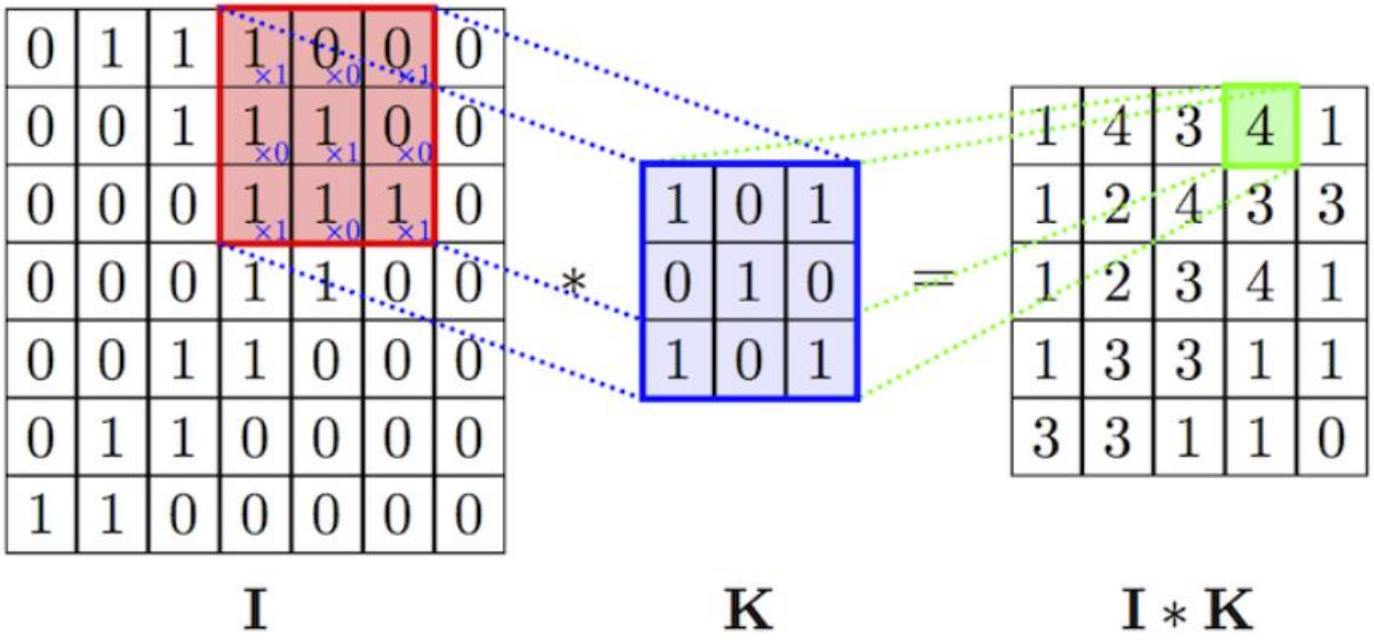
Terminology

- So far time (window slide) moved 1 unit (position) at a time. Could be any number of cells and even could be a function of another variable, e.g. we can move quicker around the ends and slower towards the center of the vector.
- The parameter that defines time speed (slide of a window between taking inputs) is called the *stride* of the layer
- 2D convolutional layers are called *planar*, 3D *spatial*, 4D *hyperspatial c*onvolutional layer
- Inputs can also be viewed as n-dimensional tensors
- The 'window' is called *receptive field.*
- The weight tensor is called a *filter* or *kernel*
- Filter sometimes include implied bias i.e. the result of it application is $I * K(t, r) + b$ for each t, r

Illustration of CNN idea (continued)

Example:

- Example of 2D convolution with 7×7 image I and 3×3 kernel K



Feature Maps

- Use many filters instead of one. Why? Recall that while optimizing a neuron we could find local optimum instead of global. So by initializing them differently we could get to a better local or global optimum!
  - For example for 10×10 image with 3 color channels we could use 3×3 filters to compute 5 parallel convolutional layers per same input in each color. So we would construct 15 matrices (images) of 8×8 dimension. We reduced image but made it deeper
  - In each neuron we would randomly initialize 10 parameters (9 weights+ bias), so there is a good chance that we get different results in each layer
- Outputs of parallel convolutional layers are called *feature maps.*
- It is enough to have 1 good feature map to drastically improve performance of CNN

Main Idea as Consequence of Convolution

- Sparse connectivity because we are creating a feature from a region in the input volume of the size of the kernel.
  - One neuron only takes inputs of the size of a filter: we are trying to explore smaller regions of the image to find shapes.
  - Shared weights – we apply the same kernel across entire picture. Will there be problems with updates in training?
- If we put together one convolution layer after another then previous layer in effect increases the receptive filed of the next layer.

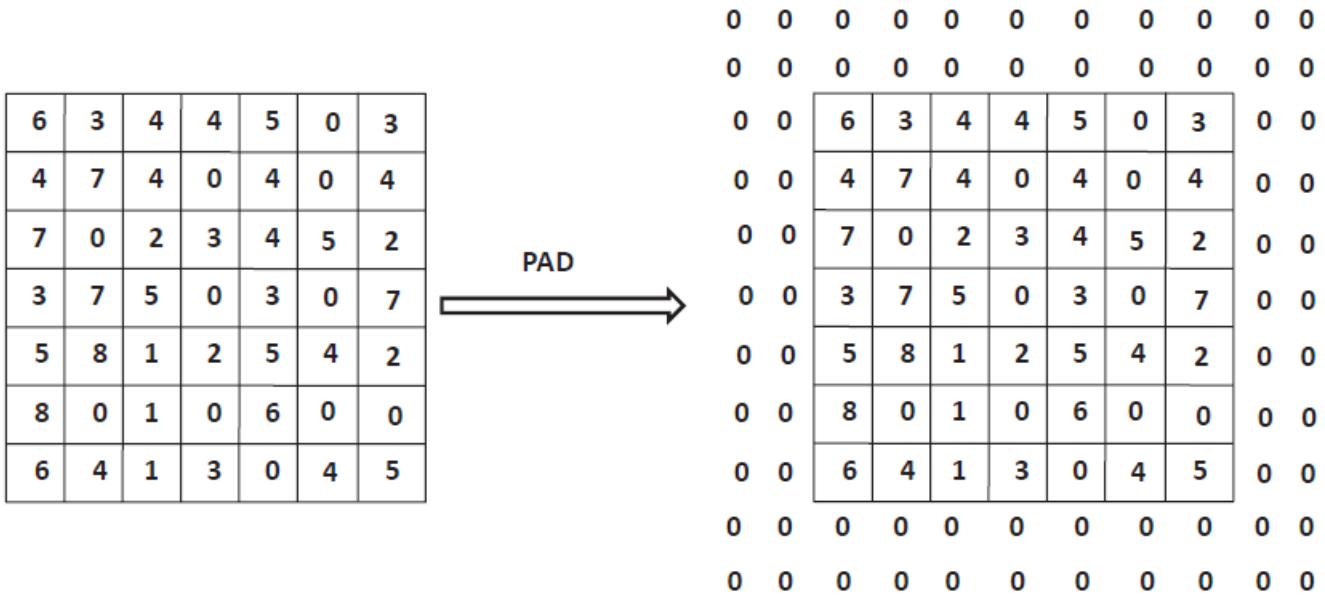
Example: Suppose we put 3×3 filters successively in three layers then:

- activations in the first, second, and third hidden layers capture pixel regions of size $3 \times 3, 5 \times 5$, and 7×7 respectively, in the *original input image*.

Padding

- We'd like to maintain picture size through the layers. If filter size of the layer q is $F_q \times F_q$ (most typical square filter) then we need to put a picture into a 'picture frame' of $\left\lfloor \frac{F_q}{2} \right\rfloor$ zeros.

Example: Suppose we put 3×3 filter on a 7×7 picture then the padding is:



Pooling

Between convolutional layers can put a pool layer:

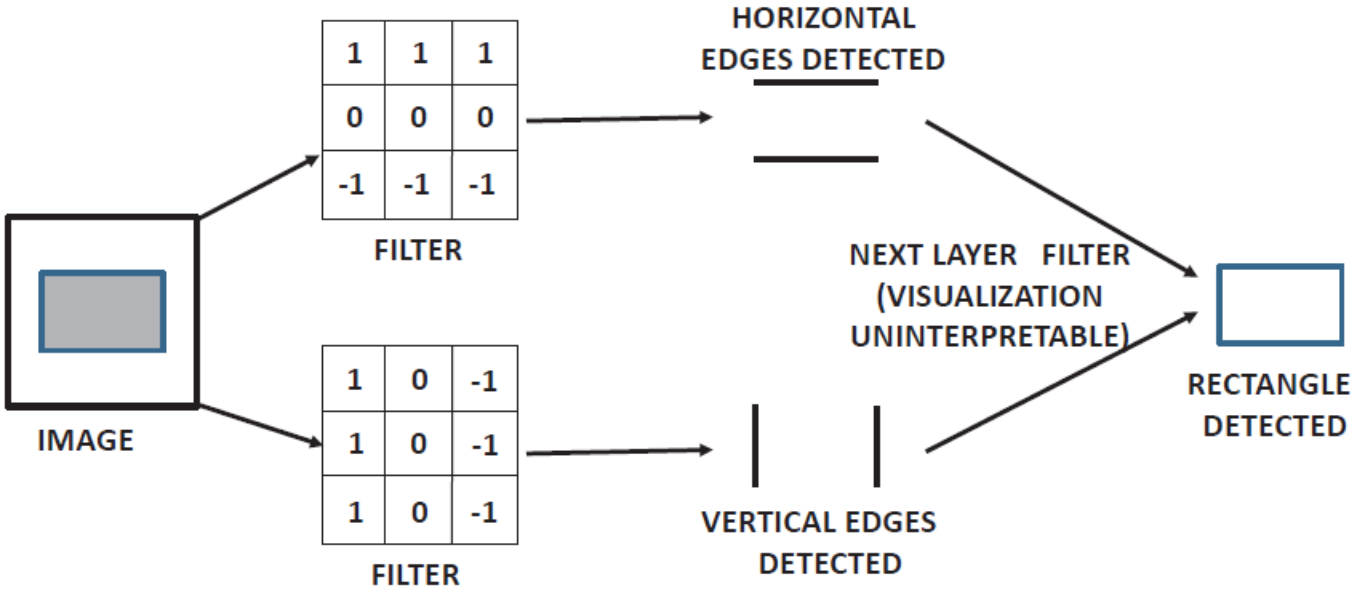
- Divide an image (feature map) into square $n \times n$ non-overlapping subareas. n is called *pool size.*
- For each pool pick a the pixel with the maximal value.
- Compose these pixels into a new image, with the same order as the original image.
- A 2×2 max-pooling layer produces an image that is half the size of the original image
- Of course, instead of the maximum, a different pixel selection or creation can be devised, such as the average of the four pixels, the minimum, and so on.

Activation

- Use of ReLU is a straightforward one-to-one operation.
- The number of feature maps and spatial footprint size is retained.
- ReLU is often stuck at the end of a convolution operation and not shown in architectural diagrams.
- Sometimes is treated as separate layer

Feature Development

- The early layers detect primitive features and later layers complex ones.
- Successive layers put together primitive features to create more complex features.
- During training the filters are learned to identify relevant shapes
- Complex features represent regularities in the data, that are valuable for features.



Fully Connected layer

The output from the convolutional layers represents high-level features in the data. Fully-connected layer is a way of learning non-linear combinations of features learned in convolutional layers.

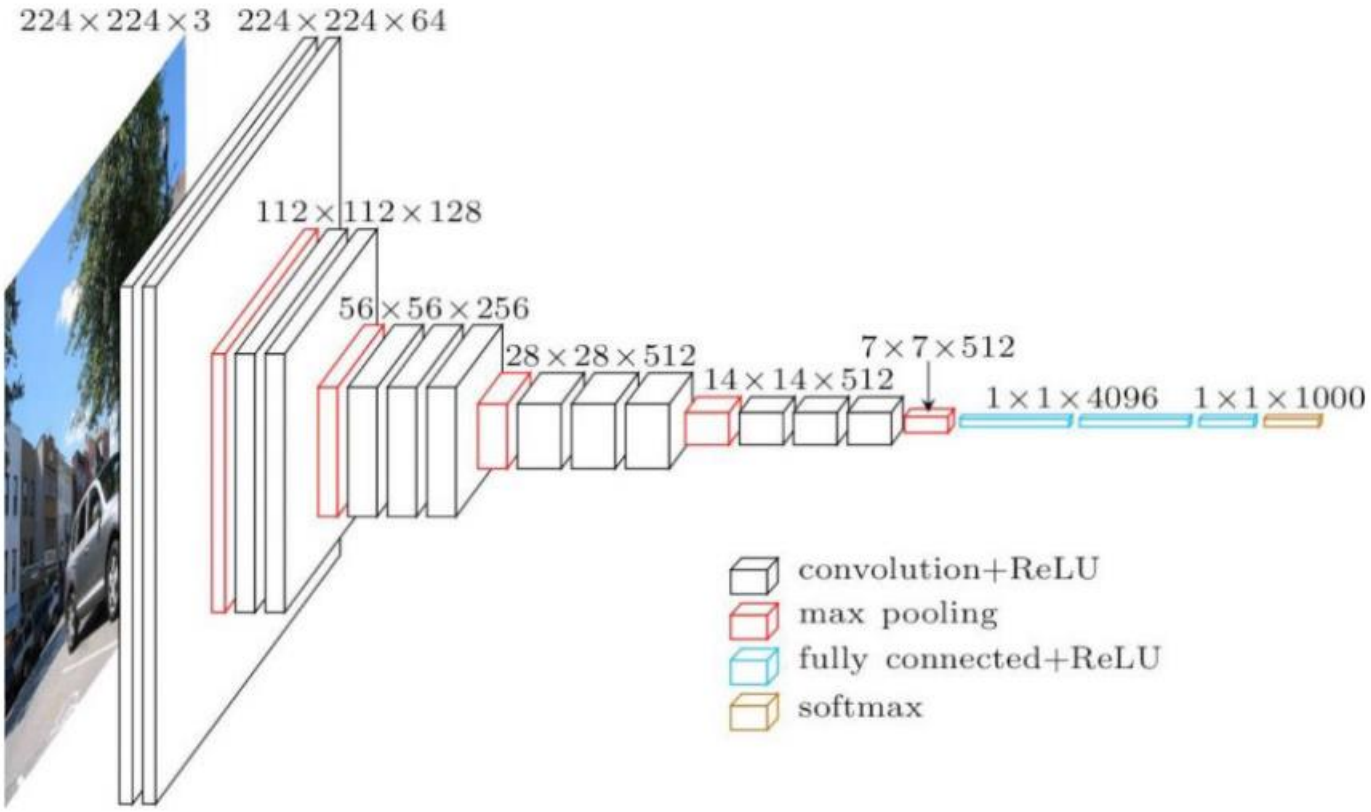
- Each feature in the final spatial layer is connected to each hidden neuron in the first fully connected layer.
- Fully connected layers function in exactly the same way as a traditional feed-forward network.
- One or more fully connected layers are used to increase the power of the computations towards the end.
- The connections among these layers are exactly the same as in standard feed-forward network.
- The vast majority of parameters lie in the fully connected layers.

Patter of Network Construction and Softmax

- The convolution, pooling, and ReLU layers are typically interleaved in a neural network in order to increase the expressive power of the network. The ReLU layers often follow the convolutional layers.
- After a few convolutional layers a pooling layer follows.
- Few repetitions of this sequence is ended with 1 or more fully connected layers. In last layer is Softmax.
- Softmax if to do classification that is done into after feature extraction. We need to classify the picture into classes, e.g. dogs, cats landscapes, portraits, etc.
- Typical pattern is

CRCRPCRCRPCRCRPFS

Typical Network Architecture



Reading

- 8.2.1-8.2.7